Figure 1

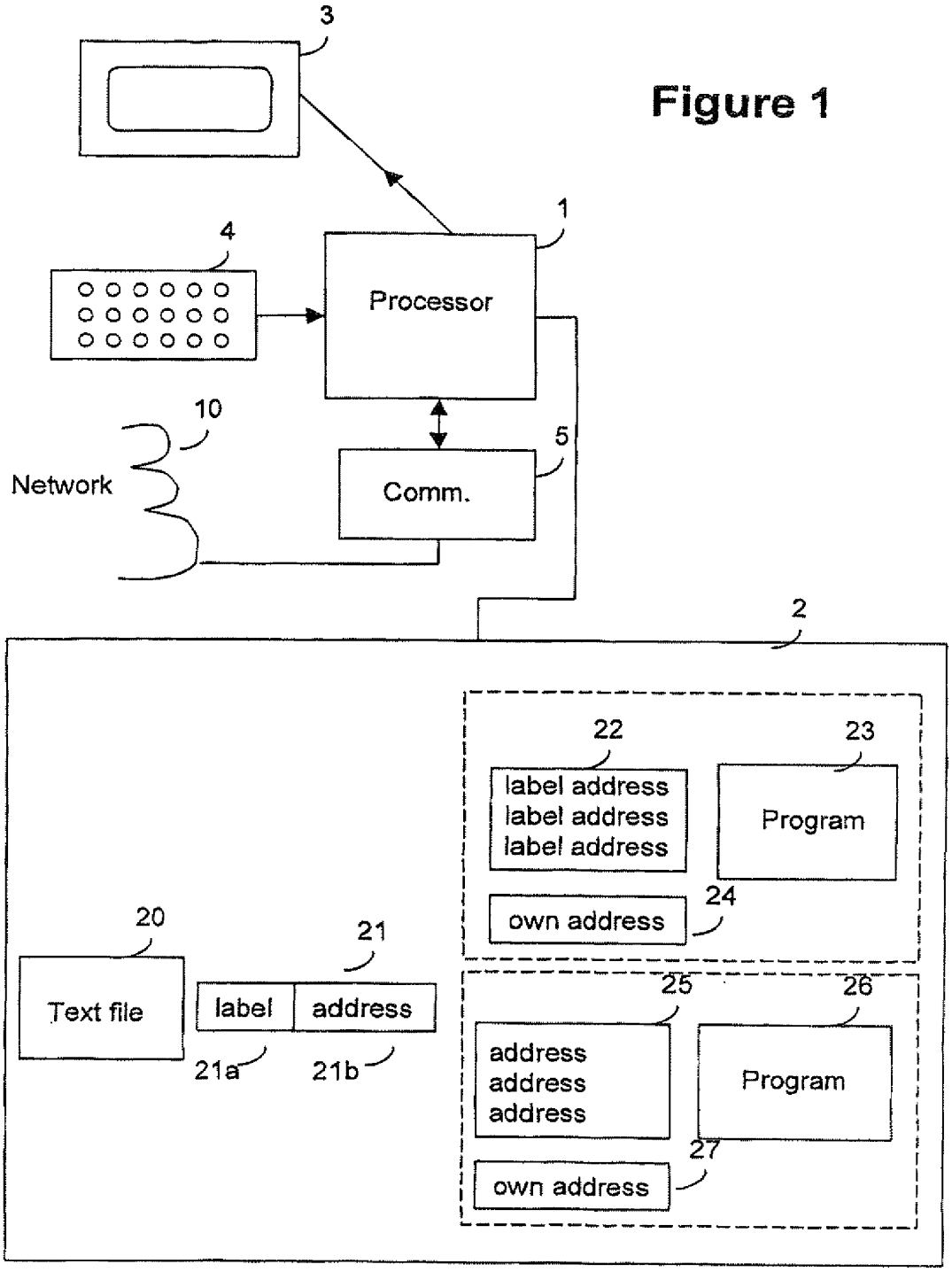
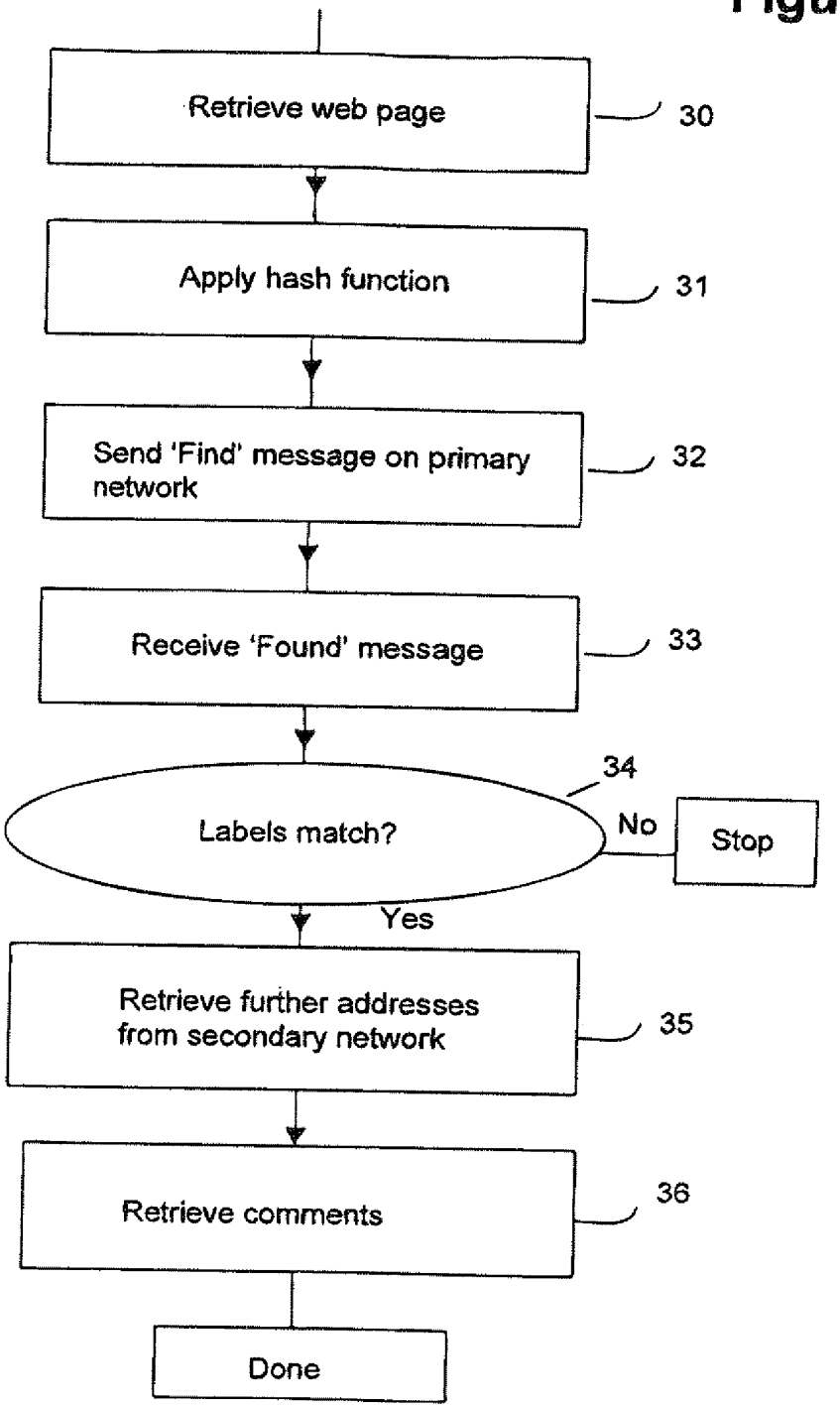
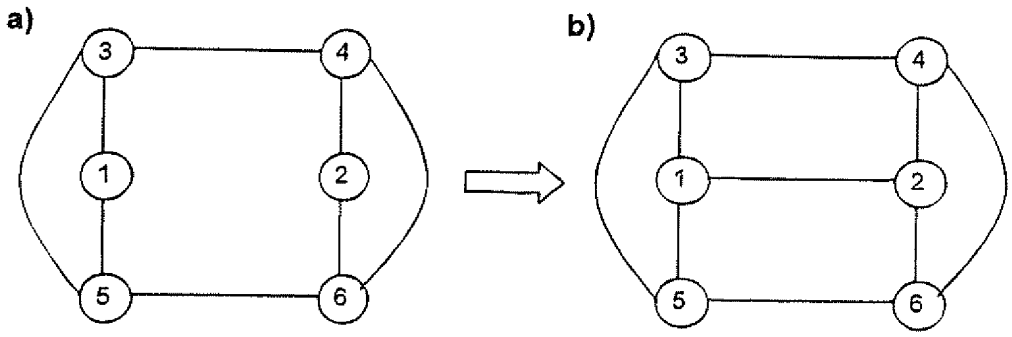
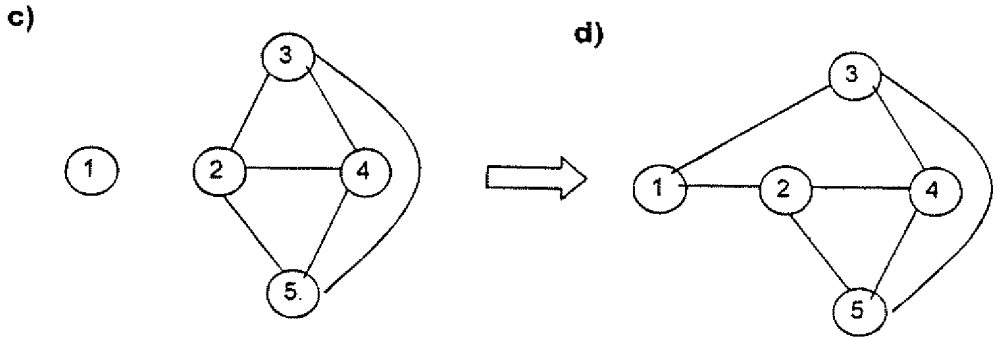

3
1
4
Processor
10
5
Network
Comm.
2
22
23
label address
label address
label address
Program
24
own address
20
21
Text file
label
address
21a
21b
25
26
address
address
address
Program
27
own address

Figure 1A


Retrieve web page
30
Apply hash function
31
Send 'Find' message on primary network
32
Receive 'Found' message
33
34
Labels match?
No
Stop
Yes
Retrieve further addresses from secondary network
35
Retrieve comments
36
Done



How a new link is created between two nodes (Node 1 and 2) when both nodes can still create a new link. a) The network before Node 1 received a Notify message about Node 2, and b) The network after Node 1 successfully created a new link in response to the Notify message.



How a node (Node 1) inserts itself in an existing link (between Node 2 and Node 3) because Node 2 already had the maximum number of links. c) The network before Node 1 received a Notify message about Node 2, and d) The network after Node 1 successfully changed an existing link into two new ones in response to the Notify message.

Figure 2

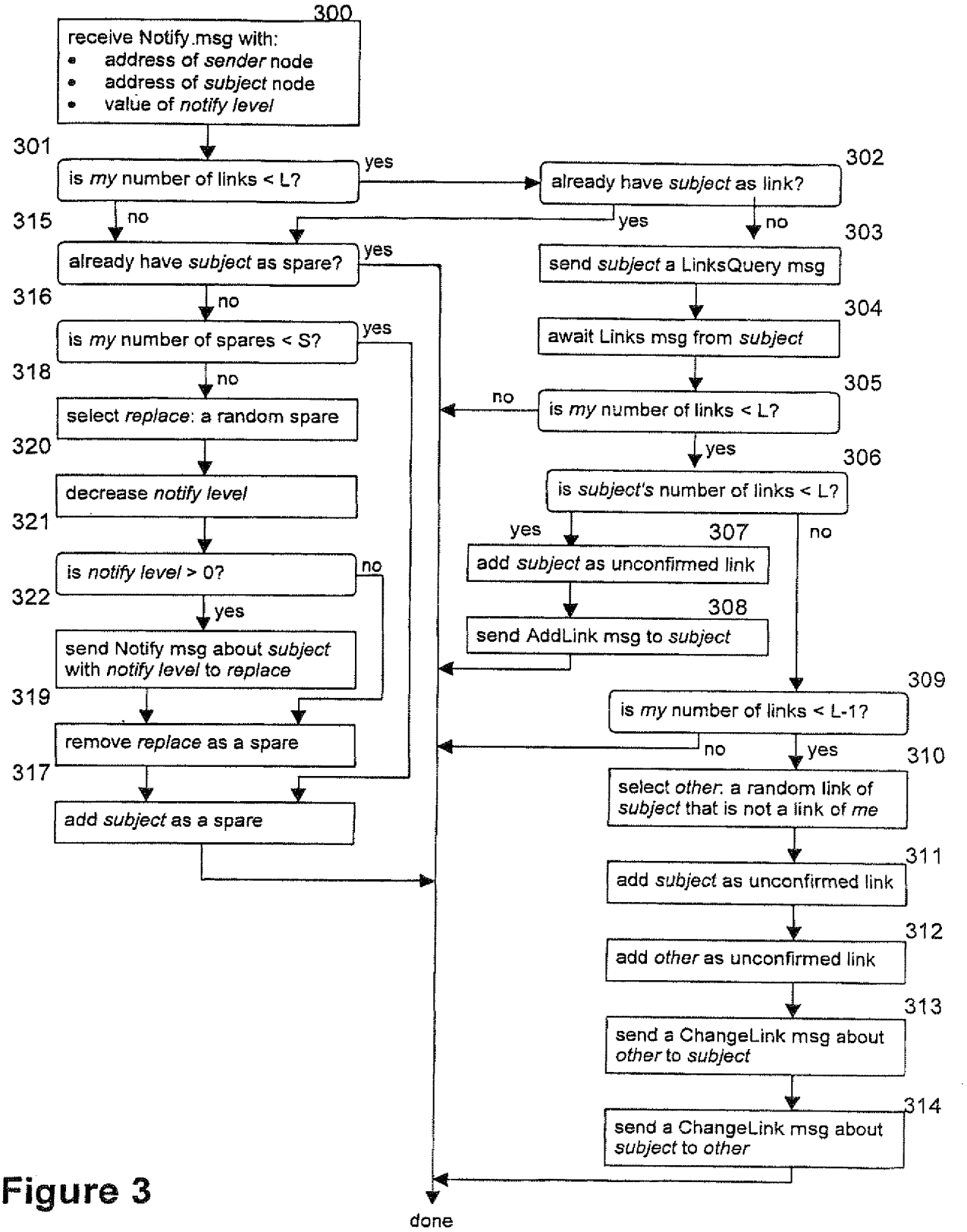

300
receive Notify.msg with:
• address of sender node
• address of subject node
• value of notify level
301
is my number of links < L?
yes
302
already have subject as link?
no
315
yes
no
303
already have subject as spare?
yes
send subject a LinksQuery msg
316
no
304
is my number of spares < S?
yes
await Links msg from subject
318
no
305
no
is my number of links < L?
select replace: a random spare
yes
320
306
decrease notify level
is subject's number of links < L?
321
yes
307
no
is notify level > 0?
no
add subject as unconfirmed link
322
yes
308
send Notify msg about subject with notify level to replace
send AddLink msg to subject
309
319
is my number of links < L-1?
remove replace as a spare
no
yes
310
317
select other: a random link of subject that is not a link of me
add subject as a spare
311
add subject as unconfirmed link
312
add other as unconfirmed link
313
send a ChangeLink msg about other to subject
314
send a ChangeLink msg about subject to other
done

Figure 3

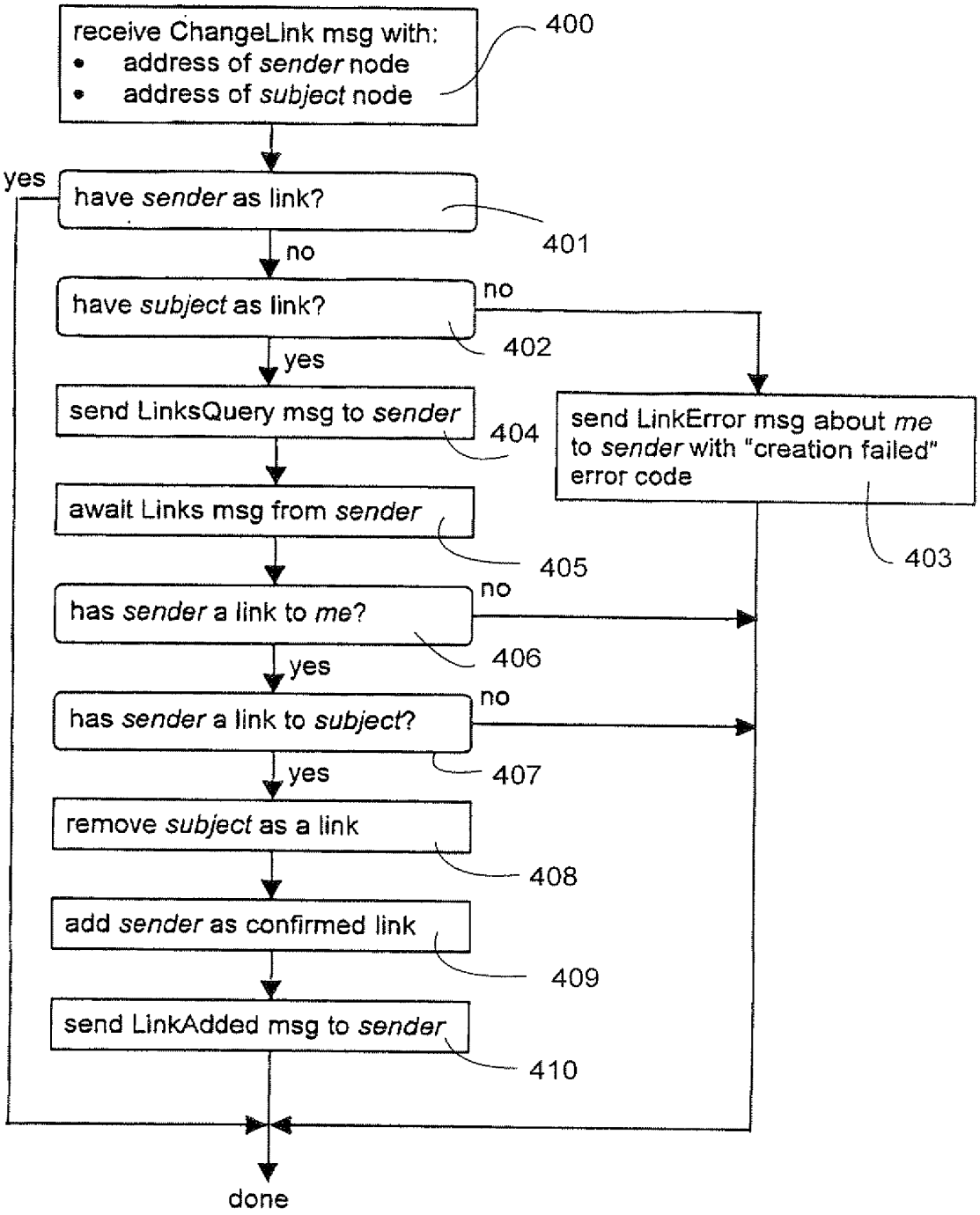

receive ChangeLink msg with:
• address of *sender* node
• address of *subject* node
400
have *sender* as link?
401
yes
no
have *subject* as link?
402
no
yes
send LinksQuery msg to *sender*
404
send LinkError msg about *me* to *sender* with "creation failed" error code
403
await Links msg from *sender*
405
has *sender* a link to *me*?
406
no
yes
has *sender* a link to *subject*?
407
no
yes
remove *subject* as a link
408
add *sender* as confirmed link
409
send LinkAdded msg to *sender*
410
done

Figure 4

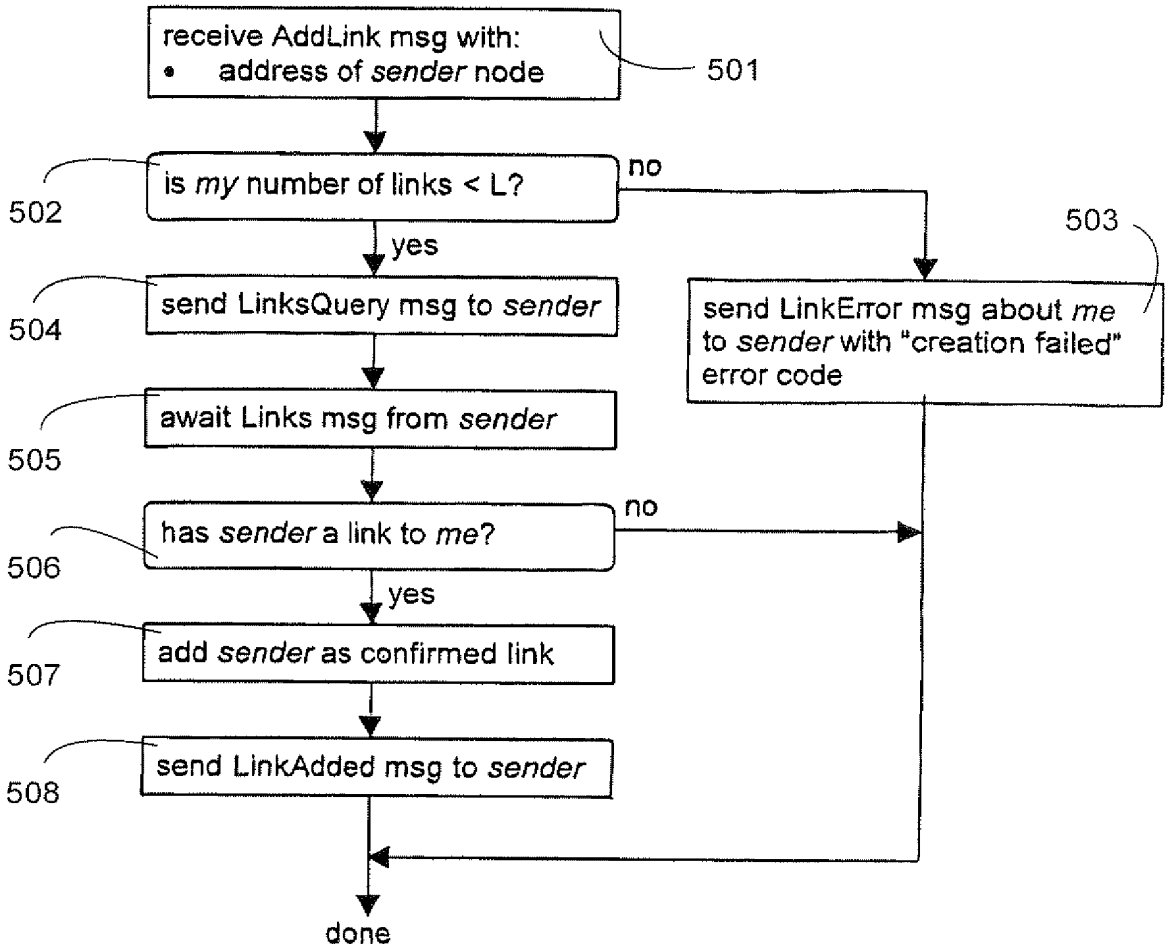


Figure 5

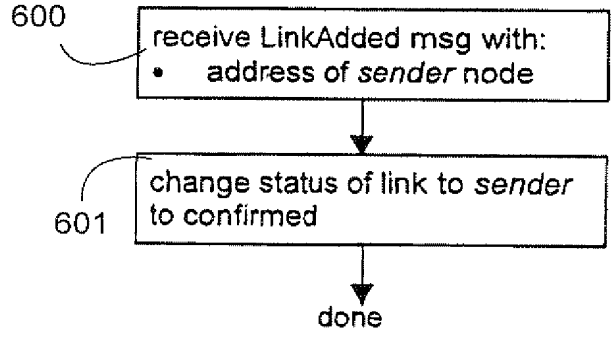


Figure 6

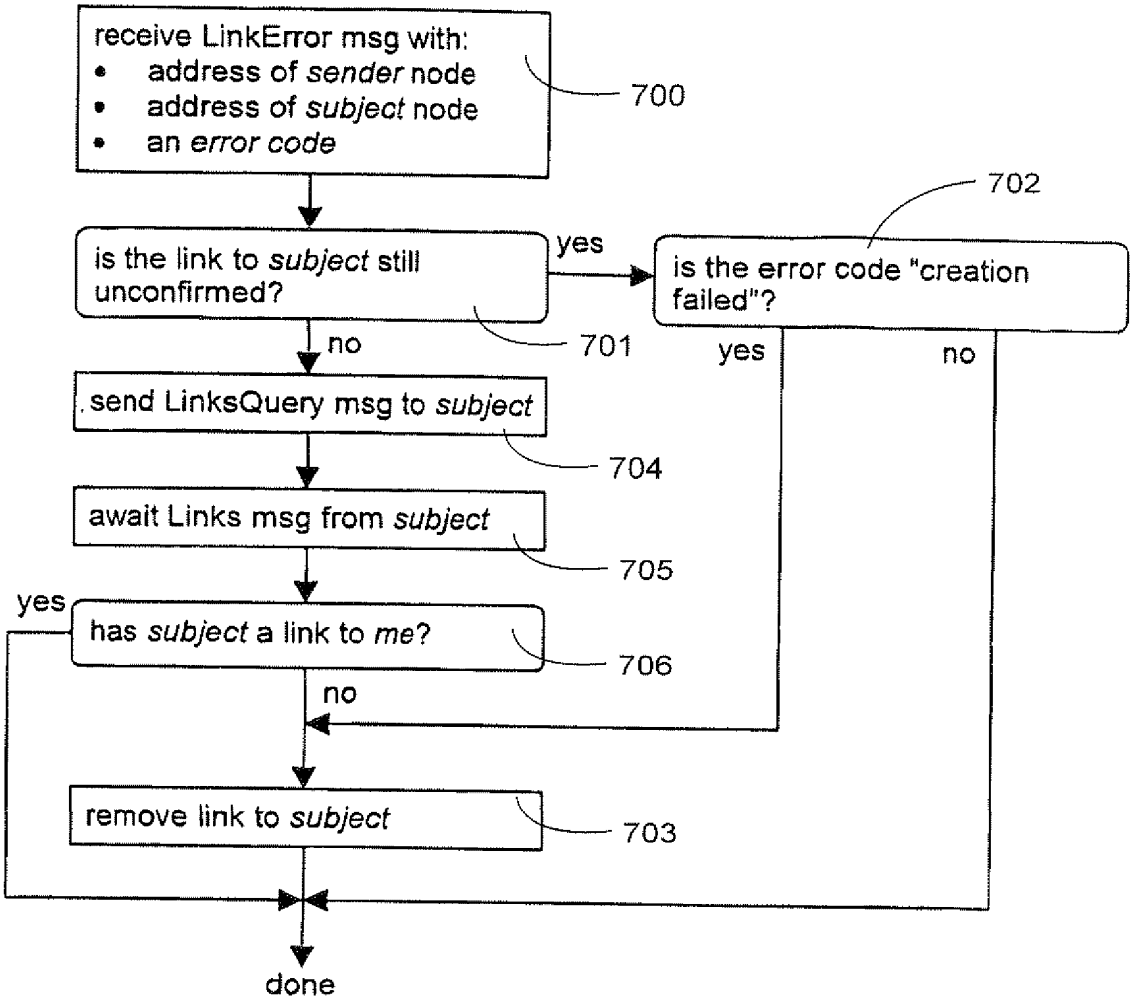

receive LinkError msg with:
• address of *sender* node
• address of *subject* node
• an *error code*
700
is the link to *subject* still unconfirmed?
yes
no
701
is the error code "creation failed"?
702
yes
no
send LinksQuery msg to *subject*
704
await Links msg from *subject*
705
yes
has *subject* a link to *me*?
706
no
remove link to *subject*
703
done

Figure 7

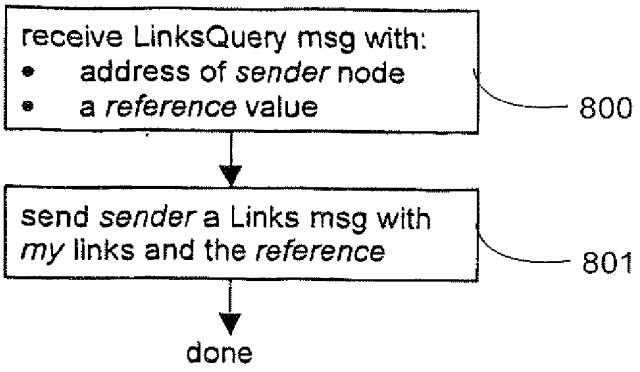


Figure 8

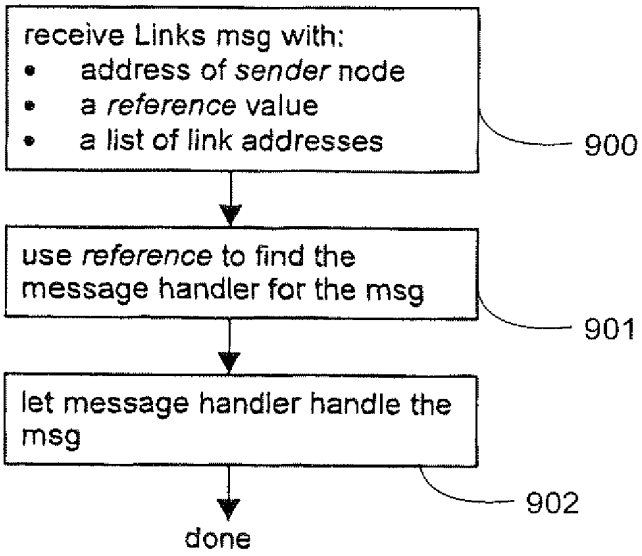


Figure 9

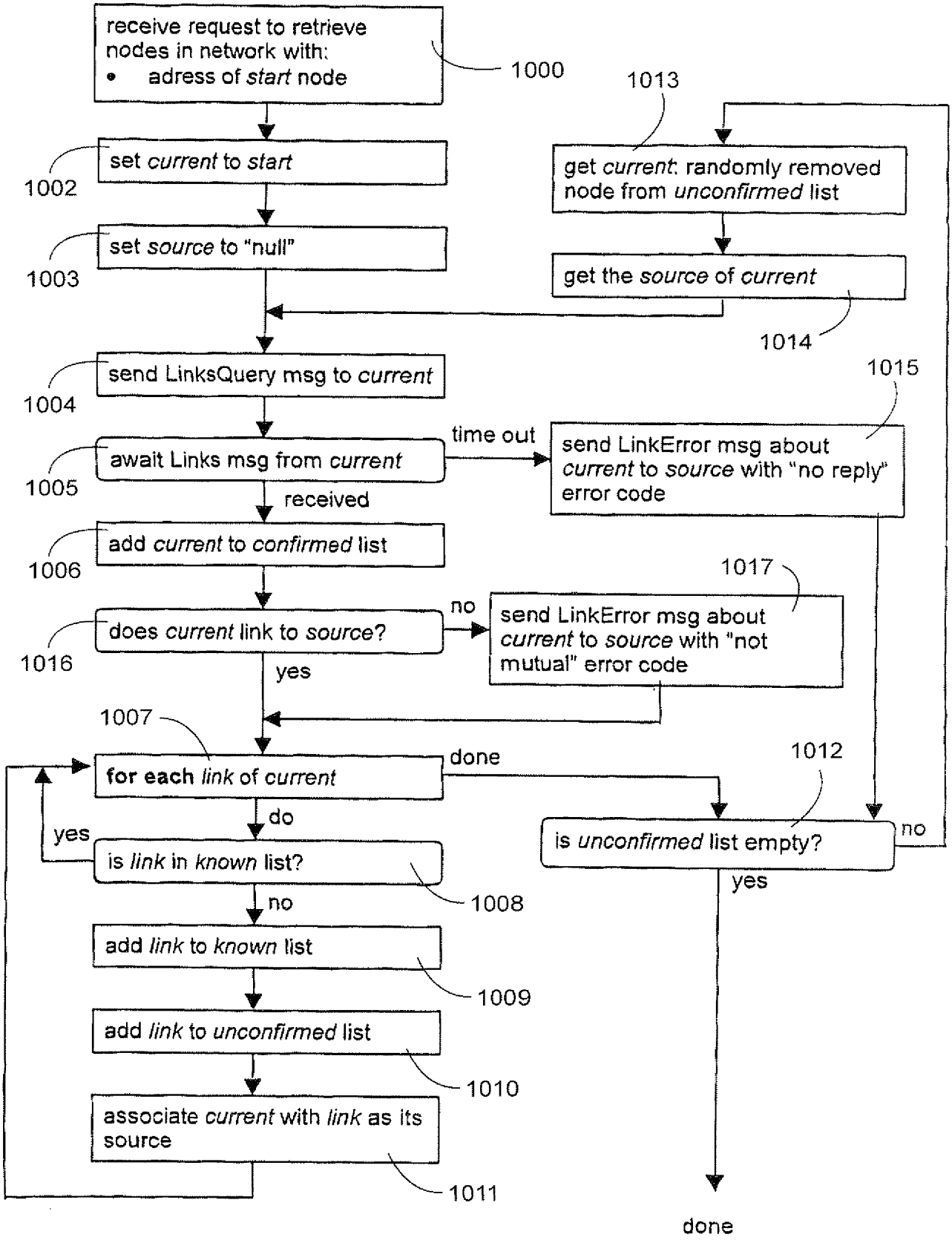

receive request to retrieve nodes in network with:
• adress of *start* node
1000
set *current* to *start*
1002
set *source* to "null"
1003
send LinksQuery msg to *current*
1004
await Links msg from *current*
1005
time out
received
add *current* to *confirmed* list
1006
does *current* link to *source*?
1016
no
yes
1007
for each *link* of *current*
done
do
yes
is *link* in *known* list?
1008
no
add *link* to *known* list
1009
add *link* to *unconfirmed* list
1010
associate *current* with *link* as its source
1011
1013
get *current*: randomly removed node from *unconfirmed* list
get the *source* of *current*
1014
1015
send LinkError msg about *current* to *source* with "no reply" error code
1017
send LinkError msg about *current* to *source* with "not mutual" error code
1012
is *unconfirmed* list empty?
no
yes
done

Figure 10

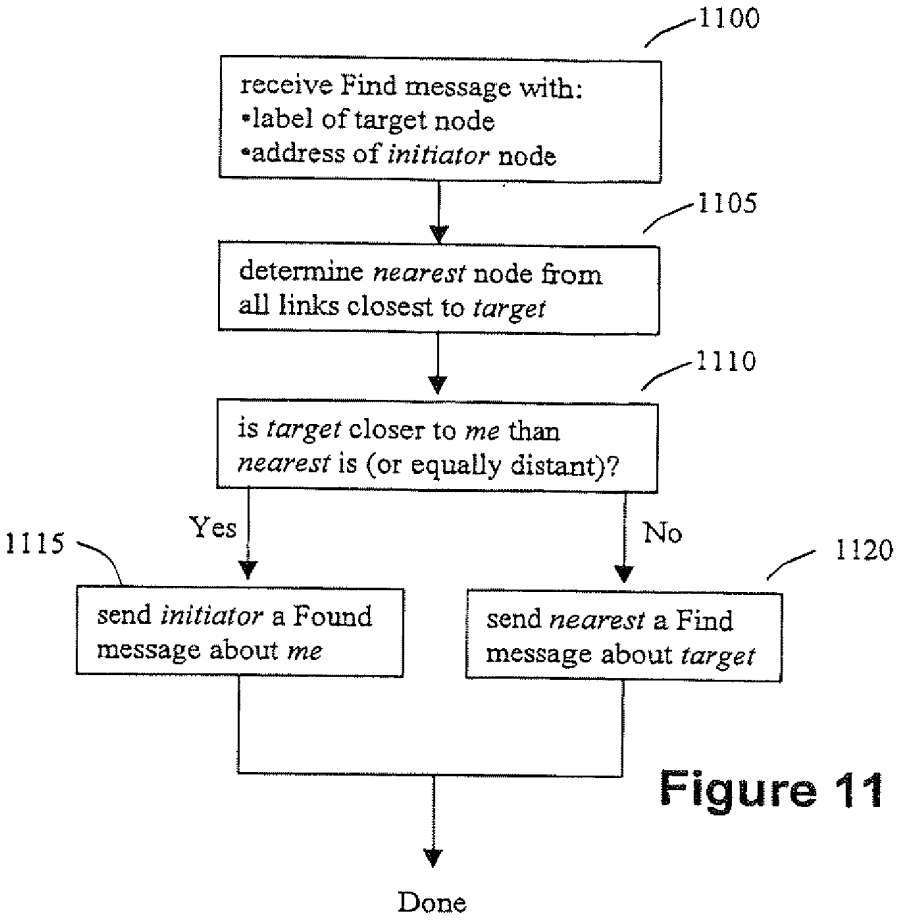


Figure 11

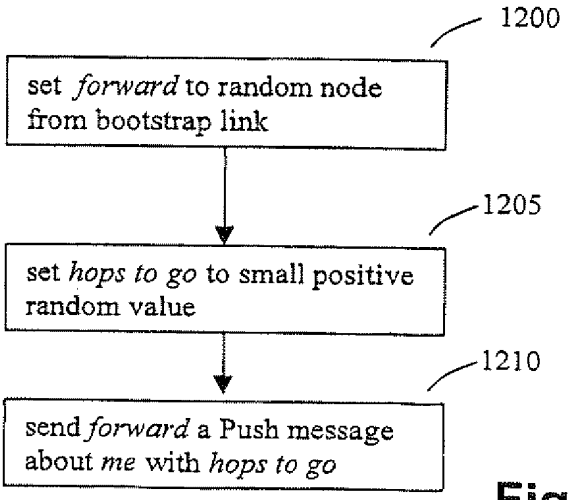


Figure 12

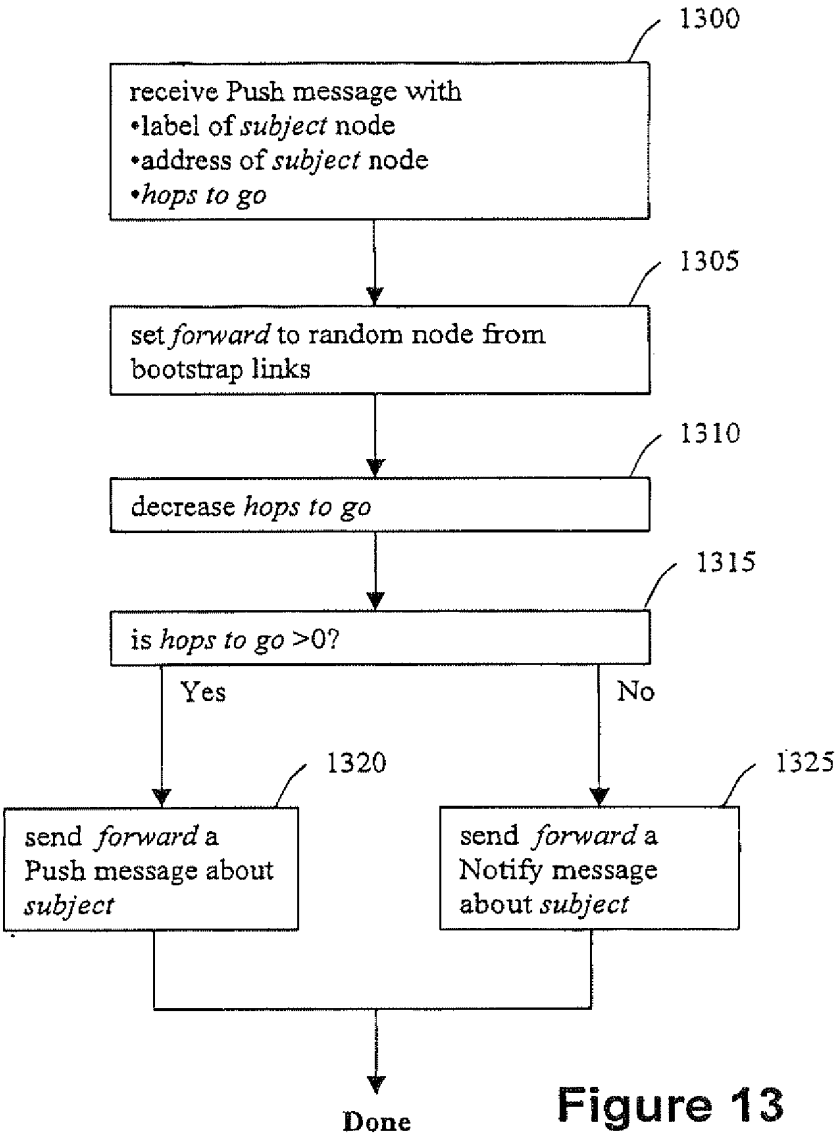

1300
receive Push message with
•label of *subject* node
•address of *subject* node
•*hops to go*
1305
set *forward* to random node from bootstrap links
1310
decrease *hops to go*
1315
is *hops to go* >0?
Yes
No
1320
send *forward* a Push message about *subject*
1325
send *forward* a Notify message about *subject*
Done

Figure 13

Figure 14

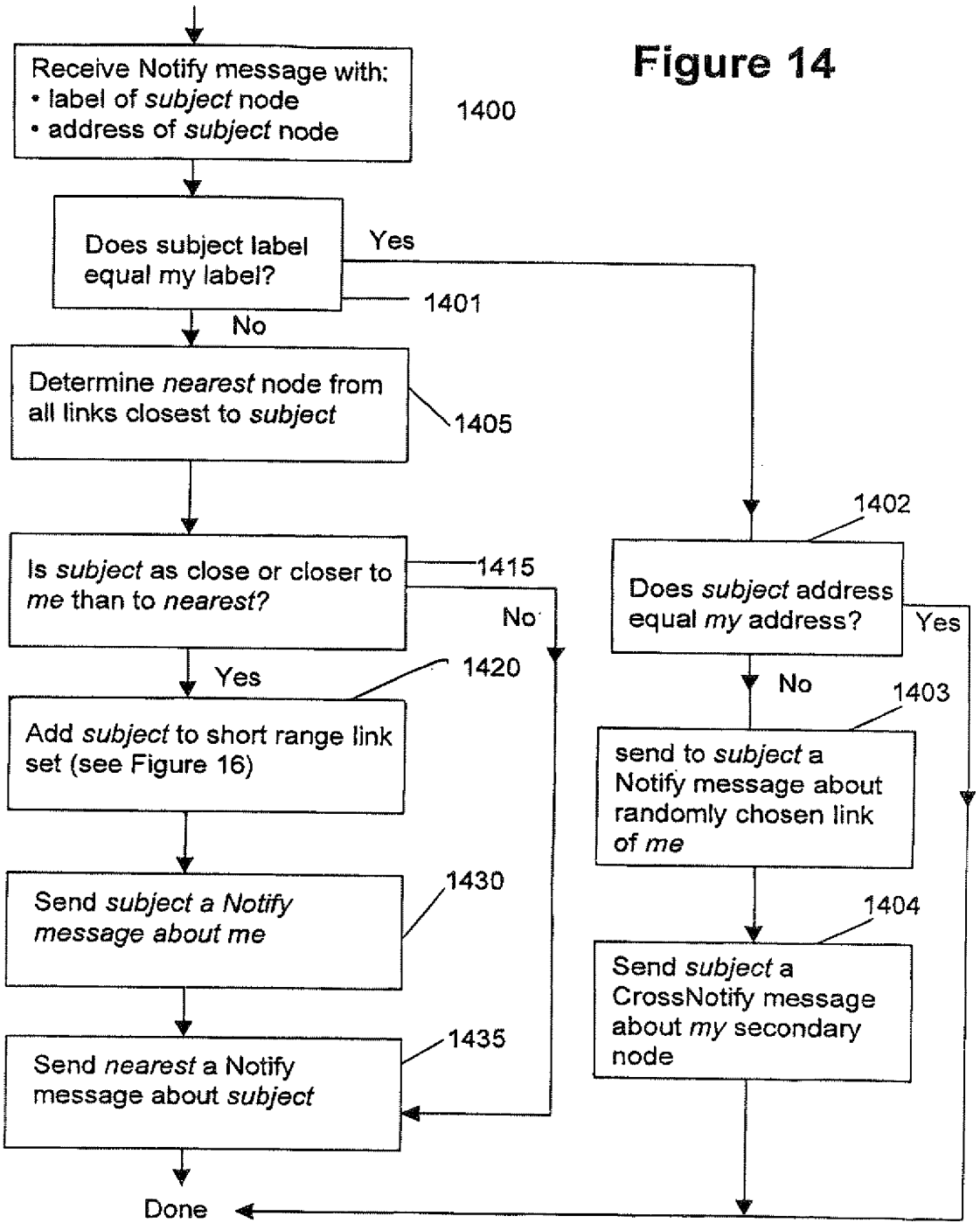
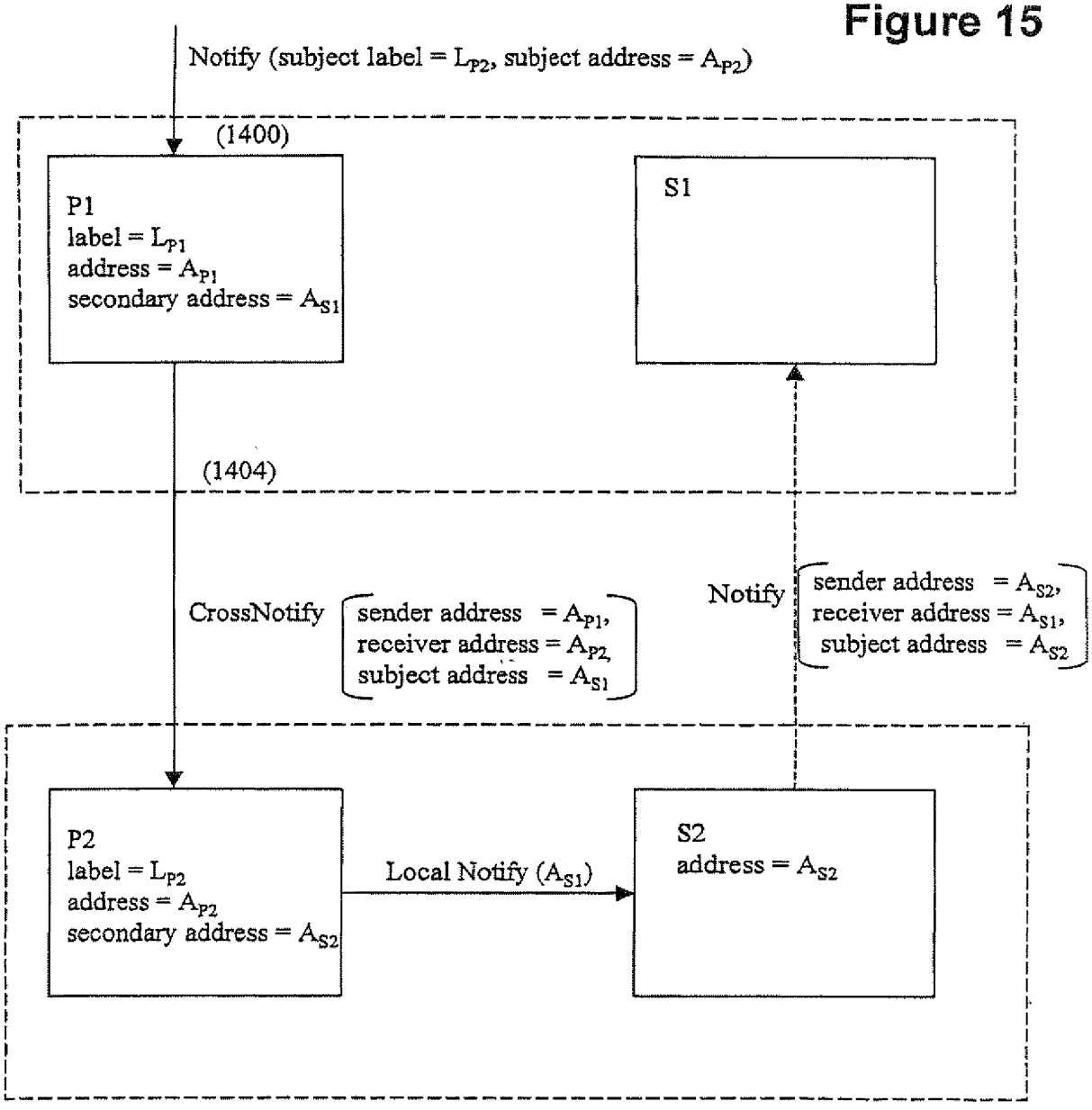

Receive Notify message with:
• label of *subject* node
• address of *subject* node
1400
Does subject label equal my label?
Yes
1401
No
Determine *nearest* node from all links closest to *subject*
1405
Is *subject* as close or closer to *me* than to *nearest*?
1415
No
Yes
1420
Add *subject* to short range link set (see Figure 16)
Send *subject a Notify message about me*
1430
Send *nearest* a Notify message about *subject*
1435
Done
1402
Does *subject* address equal *my* address?
Yes
No
1403
send to *subject* a Notify message about randomly chosen link of *me*
1404
Send *subject* a CrossNotify message about *my* secondary node

Figure 15


Notify (subject label = L_{P2}, subject address = A_{P2})
(1400)
P1
label = L_{P1}
address = A_{P1}
secondary address = A_{S1}
S1
(1404)
CrossNotify
sender address = A_{P1},
receiver address = A_{P2},
subject address = A_{S1}
Notify
sender address = A_{S2},
receiver address = A_{S1},
subject address = A_{S2}
P2
label = L_{P2}
address = A_{P2}
secondary address = A_{S2}
Local Notify (A_{S1})
S2
address = A_{S2}

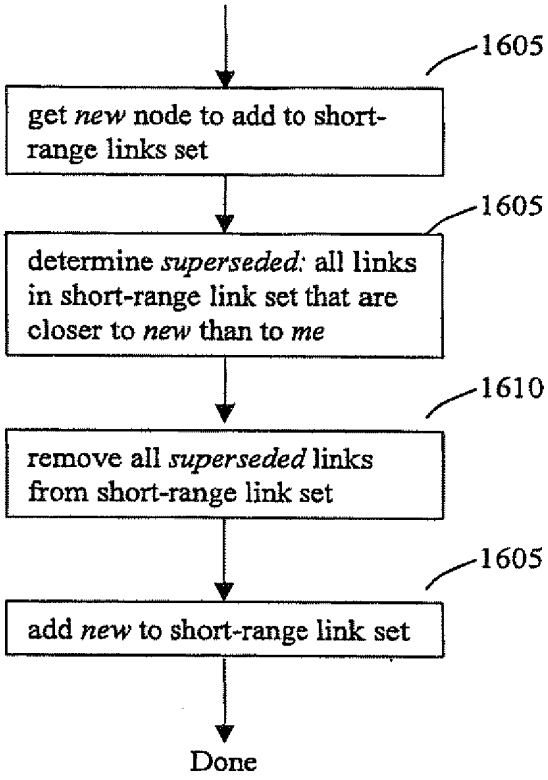

1605
get *new* node to add to short-range links set
1605
determine *superseded:* all links in short-range link set that are closer to *new* than to *me*
1610
remove all *superseded* links from short-range link set
1605
add *new* to short-range link set
Done

Figure 16

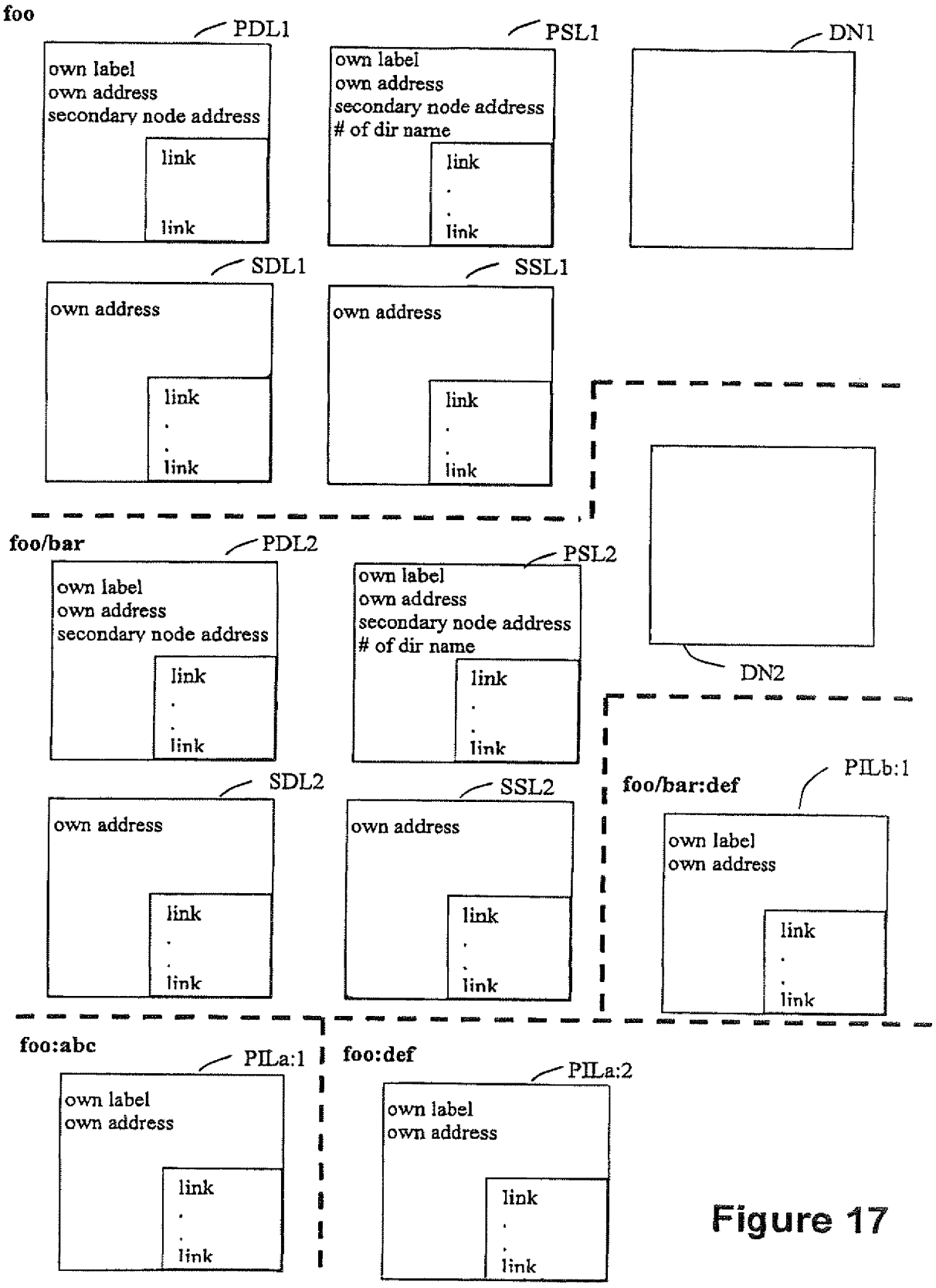

foo
PDL1
own label
own address
secondary node address
link
link
PSL1
own label
own address
secondary node address
# of dir name
link
link
DN1
SDL1
own address
link
link
SSL1
own address
link
link
foo/bar
PDL2
own label
own address
secondary node address
link
link
PSL2
own label
own address
secondary node address
# of dir name
link
link
DN2
SDL2
own address
link
link
SSL2
own address
link
link
foo/bar:def
PILb:1
own label
own address
link
link
foo:abc
PILa:1
own label
own address
link
link
foo:def
PILa:2
own label
own address
link
link

Figure 17

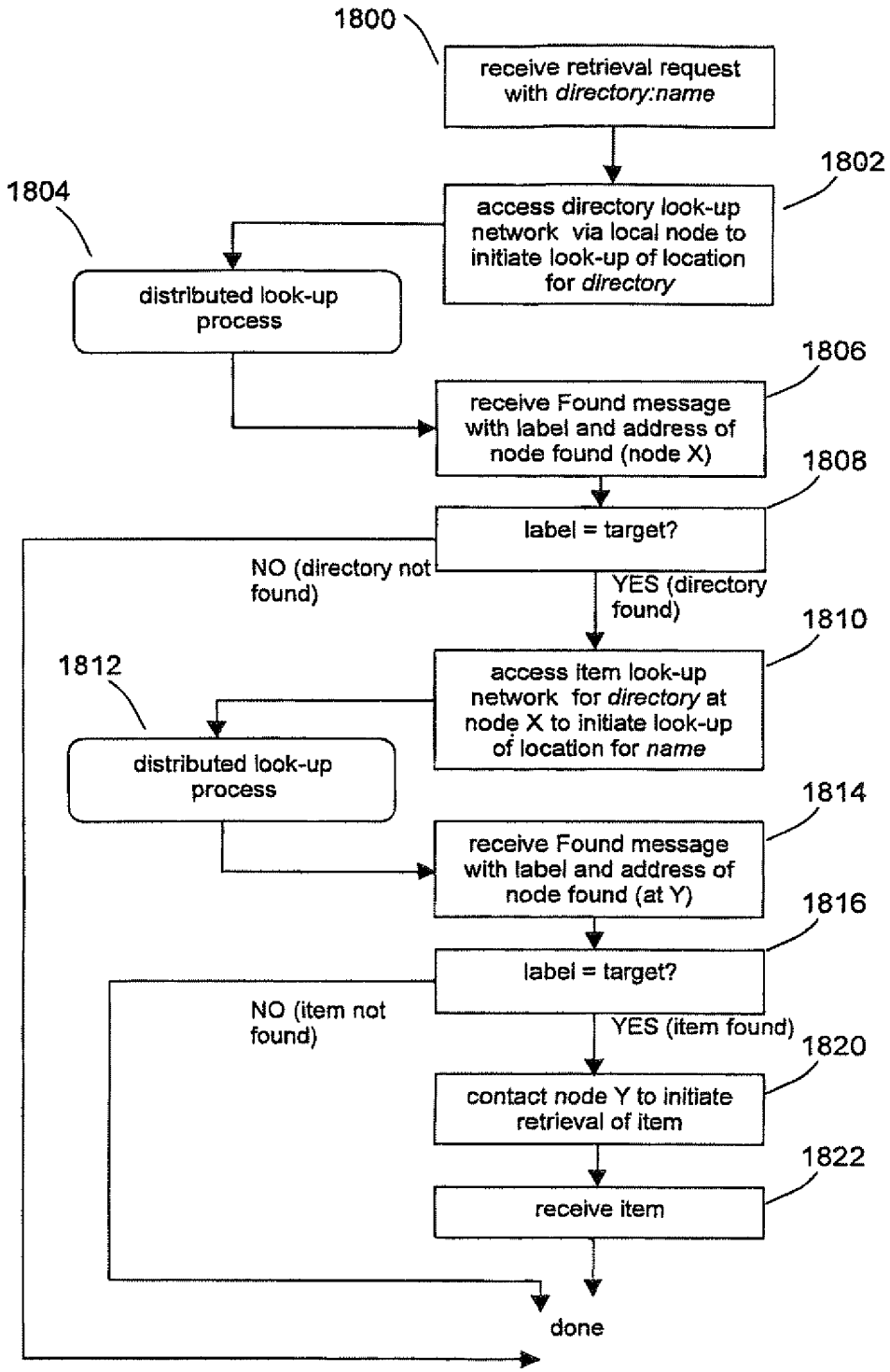

1800
receive retrieval request with directory:name
1802
access directory look-up network via local node to initiate look-up of location for directory
1804
distributed look-up process
1806
receive Found message with label and address of node found (node X)
1808
label = target?
NO (directory not found)
YES (directory found)
1810
access item look-up network for directory at node X to initiate look-up of location for name
1812
distributed look-up process
1814
receive Found message with label and address of node found (at Y)
1816
label = target?
NO (item not found)
YES (item found)
1820
contact node Y to initiate retrieval of item
1822
receive item
done

Figure 18

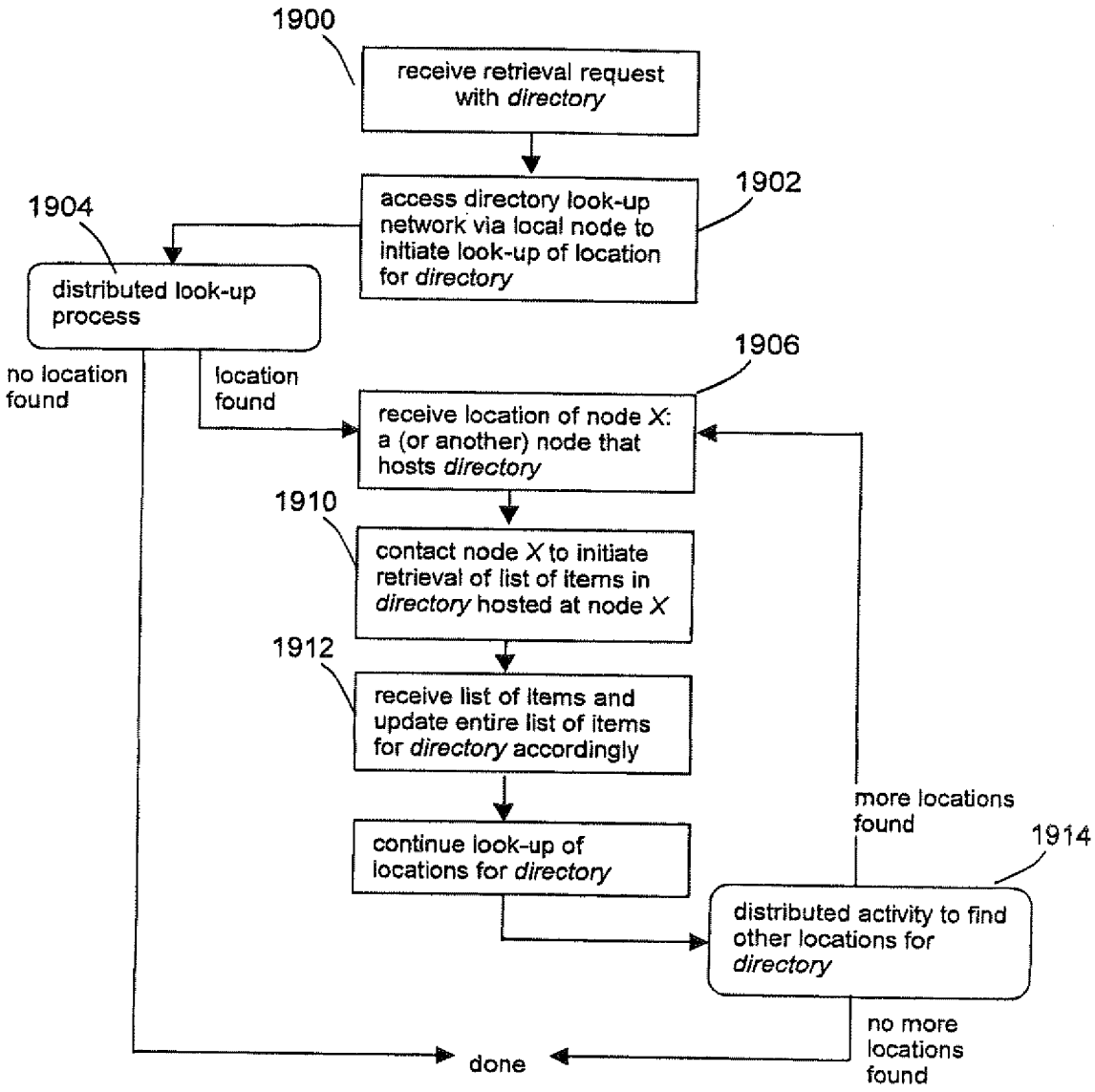

1900
receive retrieval request with *directory*
1902
access directory look-up network via local node to initiate look-up of location for *directory*
1904
distributed look-up process
no location found
location found
1906
receive location of node *X*: a (or another) node that hosts *directory*
1910
contact node *X* to initiate retrieval of list of items in *directory* hosted at node *X*
1912
receive list of items and update entire list of items for *directory* accordingly
continue look-up of locations for *directory*
more locations found
1914
distributed activity to find other locations for *directory*
no more locations found
done

Figure 19

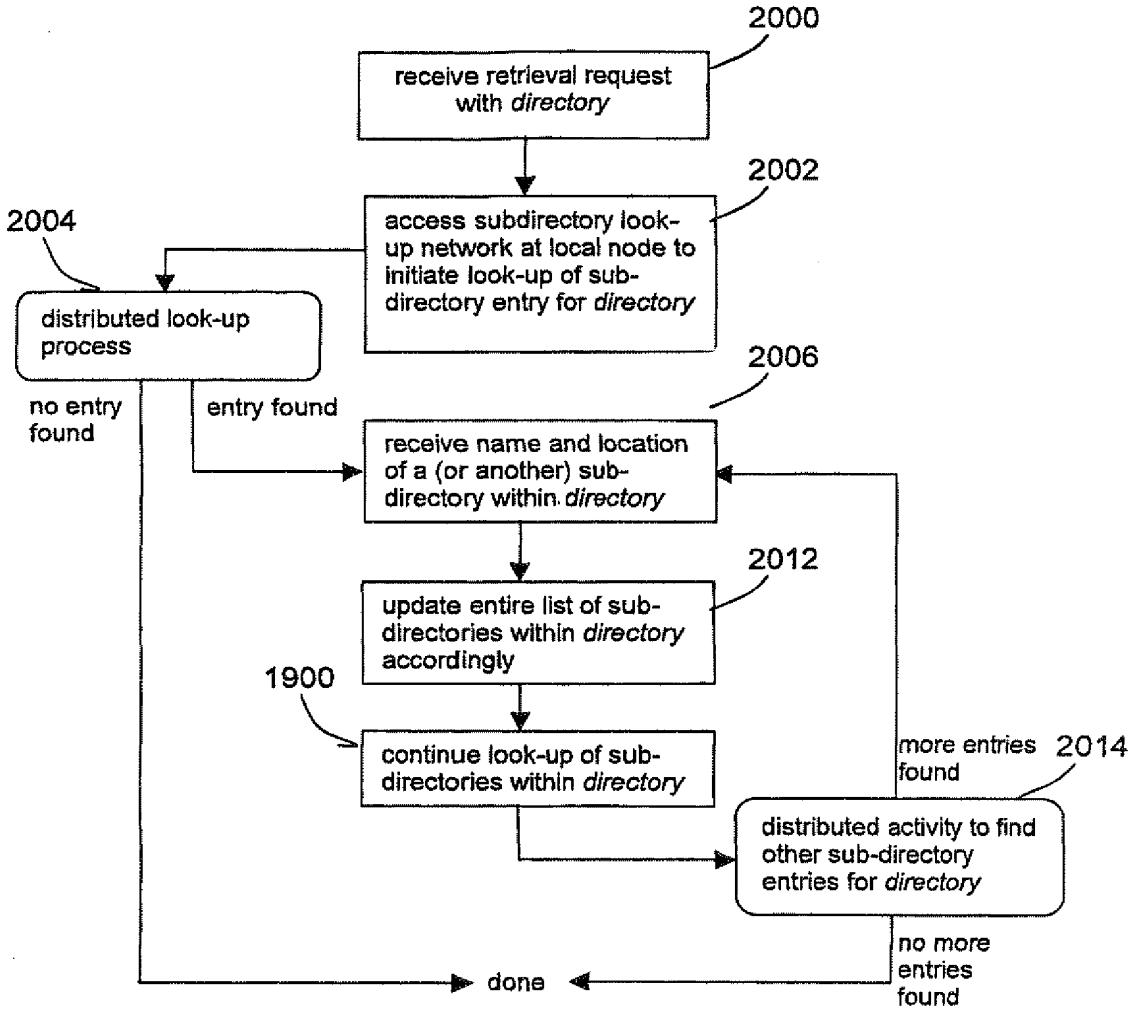

2000
receive retrieval request with *directory*
2002
access subdirectory look-up network at local node to initiate look-up of sub-directory entry for *directory*
2004
distributed look-up process
no entry found
entry found
2006
receive name and location of a (or another) sub-directory within *directory*
2012
update entire list of sub-directories within *directory* accordingly
1900
continue look-up of sub-directories within *directory*
more entries found
2014
distributed activity to find other sub-directory entries for *directory*
no more entries found
done

Figure 20